Marathwada Mitra Mandal's
COLLEGE OF ENGINEERING
Karvenagar, PUNE – 411 052
Accredited with "A++" Grade by NAAC // Accredited by NBA (Mechanical Engg. & Electrical Engg.)
Recipient of "Best College Award 2019" by SPPU // Recognized under section 2(f) and 12B of UGC Act 1956

DEPARTMENT OF INFORMATION TECHNOLOGY

LABORATORY MANUAL

BE (INFORMATION TECHNOLOGY)

(SEMESTER – I)

LABORATORY PRACTICE – IV
[DEEP LEARNING LABORATORY]

2019 Course

Prepared By
Mrs. Shraddha P. Mankar

Preface

This lab manual has been prepared to help the students in studying and analyzing various fascinating real of artificial intelligence, specifically focusing on deep learning techniques and applications.

Deep learning, a subset of machine learning inspired by the structure and function of the human brain, has revolutionized various fields such as computer vision, natural language processing, and reinforcement learning. Its ability to automatically learn representations of data through the use of neural networks has led to remarkable advancements in areas ranging from image recognition to language translation. It is structured to provide you with hands-on experience in implementing and understanding deep learning algorithms. Throughout the manual, you will find a series of lab exercises designed to guide you through fundamental concepts, methodologies, and best practices in deep learning. These exercises are structured progressively, starting from basic neural network architectures and advancing to more complex models and applications to be adaptable to various deep learning frameworks, including TensorFlow, PyTorch, and Keras. You are encouraged to use the framework of your choice, although specific examples may be provided using one of these frameworks.

As you embark on your journey through the world of deep learning, remember that experimentation and exploration are key. Don't hesitate to tweak parameters, try out different architectures, and push the boundaries of what you can achieve with deep learning. Lastly, I would like to express my gratitude to all the researchers, educators, and developers whose contributions have made deep learning accessible to a wider audience. Their dedication and innovation continue to inspire us in our pursuit of knowledge and understanding. Deep learning helps to improve organizational security, integration, compliance, and performance. Proper mapping is also done with Program Outcomes and Program Specific Outcomes.

Marathwada Mitramandal's
COLLEGE OF ENGINEERING
S.No.18, Plot No.5/3, Karvenagar, Pune-52
Accredited with "A++" Grade by NAAC | Recipient of "Best College Award 2019" by SPPU
Accredited by NBA

- Vision of the Institute

• To aspire for the welfare of society through excellence in science and technology.

Mission of the Institute

• Mould young talent for higher endeavors.
• Meet the challenges of globalization.
• Commit for social progress with values and ethics.
• Orient faculty and students for research and development.
• Emphasize excellence in all disciplines.

Department of Information Technology

Vision of the Department

• To emerge as a center of excellence in education, research and innovation in Information Technology for enrichment of society.

Mission of the Department

• To cater industry with engineers having theoretical & practical background and competent IT skills.
• To pursue advanced knowledge in the field of information technology. • To inculcate budding IT engineers with professional and interpersonal skills.

Program Educational Objectives (PEOs)

The students of Information Technology Program after passing out will possess:

1. Adequate knowledge and skills in Information Technology for implementation of complex problems with innovative approaches.
2. Inclination and technical competency towards professional growth in the field of Information Technology.
3. Ethics and value based interpersonal skills to facilitate lifelong learning and societal contributions.

Program Outcomes (POs)

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems. 2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs)

Information Technology Graduates will be able to:

1. Develop quality computer applications by applying principles of software engineering.
2. Pursue advancement in the field of data engineering.

Marathwada Mitra Mandal's
COLLEGE OF ENGINEERING
S.No.18, Plot No.5/3, Karvenagar, PUNE -52
Accredited with "A++" Grade by NAAC // Accredited by NBA Recipient of "Best College Award 2019" by SPPU // Recognized under section 2(f) and 12B of UGC Act 1956

DEPARTMENT OF INFORMATION TECHNOLOGY

List of Experiments

Academic Year: 2023-24 **Semester :** I **Course Code & Name:** Lab Practice IV (Deep Learning) **Class:** BE **Course Coordinator:** Mrs. Shraddha P.Mankar

Sr. No.	Expt. No.	Experiment Title
		As per SPPU Syllabus
1	1	Study of Deep learning Packages: Tensorflow , Keras, Theano and PyTorch. Document the distinct Features and functionality of the packages.
2	2	Implementing Feedforward neural networks with Keras and TensorFlow a. Import the necessary packages b. Load the training and testing data (MNIST/CIFAR10) c. Define the network architecture using Keras d. Train the model using SGD e. Evaluate the network f. Plot the training loss and accuracy
3	3	Build the Image classification model by dividing the model into following 4 stages: a. Loading and preprocessing the image data b. Defining the model's architecture c. Training the model d. Estimating the model's performance
4	4	Use Autoencoder to implement anomaly detection. Build the model by using: a. Import required libraries b. Upload / access the dataset c. Encoder converts it into latent representation d. Decoder networks convert it back to the original input e. Compile the models with Optimizer, Loss, and Evaluation Metrics
5	5	Implement the Continuous Bag of Words (CBOW) Model. Stages can be: a. Data preparation b. Generate training data c. Train model d. Output

6	6	Object detection using Transfer Learning of CNN architectures. a. Load in a pre trained CNN model trained on a large dataset b. Freeze parameters (weights) in model's lower convolutional layers c. Add custom classifier with several layers of trainable parameters to model d. Train classifier layers on training data available for task e. Fine-tune hyper parameters and unfreeze more layers as needed.
Virtual Lab		
NA		

Course Code: 414447
Course Name: LABORATORY PRACTICE – IV (DL) (2019 Course)

ASSIGNMENT NO.1

Title : Study of Deep learning Packages: Tensorflow, Keras, Theano and PyTorch. Document the distinct features and functionality of the packages.

Aim: Study and installation of following Deep learning Packages :

i. Tensor Flow

ii. Keras

iii. Theno

iV . PyTorch

Theory :

1) What is Deep learning?

2) What are various packages in python for supporting Machine Learning libraries and which are mainly used for Deep Learning ?

3) Compare Tensorflow / Keras/Theno and PyTorch on following points(make a table):

i. Available Functionality

ii. GUI status

iii. Versions

iv. Features

v. Compatibility with other environments.

vi. Specific Application domains.

4) Enlist the Models Datasets and pretrained models, Libraries and Extensions , Tools related to Tensorflow also discuss any two case studies like (PayPal, Intel, Etc.) related to Tensor Flow. [Ref:https://www.tensorflow.org/about]

5) Explain the Keras Ecosystem. (keras tuner,kerasNLP,kerasCV,Autokeras and

Model Optimization.) Also explain following concepts related to keras :

1. Developing

sequential Model 2. Training and validation using the inbuilt functions 3. Parameter Optimization. [Ref: https://keras.io/]

6) Explain a simple Theano program.

7) Explain PyTorch Tensors . And also explain Uber' s Pyro, Tesala

Autopilot.[https://pytorch.org/]

Steps/ Algorithm

Installation of Tensorflow On Ubntu:

1. Install the Python Development Environment:

You need to download Python, the PIP package, and a virtual environment. If these packages are already installed, you can skip this step.

You can download and install what is needed by visiting the following links:

https://www.python.org/

https://pip.pypa.io/en/stable/installing/https://docs.python.org/3/library/venv.html

To install these packages, run the following commands in the terminal: sudo apt update

sudo apt install python3-dev python3-pip python3-venv

2. Create a Virtual Environment

Navigate to the directory where you want to store your Python 3.0 virtual environment. It can be in your home directory, or any other directory where your user can read and write permissions.

mkdir tensorflow_files

cd tensorflow_files

Now, you are inside the directory. Run the following command to create a virtual environment:

python3 -m venv virtualenv

The command above creates a directory named virtualenv. It contains a copy of the Python binary, the PIP package manager, the standard Python library, and other supporting files.

3. Activate the Virtual Environment

source virtualenv/bin/activate

Once the environment is activated, the virtual environment' s bin directory will be added to the beginning of the $PATH variable. Your shell' s prompt will alter, and it will show the name of the virtual environment you are currently using, i.e. virtualenv.

4. Update PIP

pip install --upgrade pip

5. Install TensorFlow

The virtual environment is activated, and it' s up and running. Now, it' s time to install the TensorFlow package.

pip install -- upgrade TensorFlow

Installation of Keras on Ubntu :

Prerequisite : Python version 3.5 or above.

STEP 1: Install and Update Python3 and Pip

Skip this step if you already have Python3 and Pip on your machine.

sudo apt install python3 python3.pipsudo pip3 install – – upgrade

pip STEP 2: Upgrade Setuptools

pip3 install – – upgrade setuptools

STEP 3: Install TensorFlow

pip3 install tensorflow

Verify the installation was successful by checking the software package information:

pip3 show tensorflow

STEP 4: Install Keras

pip3 install keras

Verify the installation by displaying the package information:

pip3 show keras

[https://phoenixnap.com/kb/how-to-install-keras-on-linux]

Installation of Theano on Ubuntu:

Step 1: First of all, we will install Python3 on our Linux Machine. Use the

following command in the terminal to install Python3.

sudo apt-get install python3

Step 2: Now, install the pip module

udo apt install python3-pip

Step 3: Now, install the Theano

Verifying Theano package Installation on Linux using PIP

python3 -m pip show theano

Installation of PyTorch

First, check if you are using python' s latest version or not.Because PyGame requires

python 3.7 or a higher version

python3 – version

pip3 – version

pip3 install torch==1.8.1+cpu torchvision==0.9.1+cpu torchaudio==0.8.1 -f

https://download.pytorch.org/whl/torch_stable.html

[Ref : https://www.geeksforgeeks.org/install-pytorch-on-linux/]

Python Libraries and functions required;

1. Tensorflow,kerasnumpy :

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy stands for Numerical Python. To import numpy use import numpy as np pandas: pandas is a fast, powerful, flexible and easy to use open source data analysisand manipulation tool, built on top of the Python programming language. To importpandas use import pandas as pd sklearn : Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learningand statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib. For importing train_test_split use from sklearn.model_selection import train_test_split

2. For TheaonRequirements:

Python3

Python3-pip

NumPy

SciPy

BLAS

Sample Code with comments

1. Tensorflow Test program:2. Keras Test Program:

```
from tensorflow import keras
from keras import datasets
# Load MNIST data
#(train_images, train_labels), (test_images, test_labels) =
datasets.mnist.load_data() # Check the dataset loaded
#train_images.shape, test_images.shape
```

3. Theano test program

```
# Python program showing
# addition of two scalars
# Addition of two scalars
import numpy
import theano.tensor as T
from theano import function
# Declaring two variables
x = T.dscalar('x')
y = T.dscalar('y')
# Summing up the two numbers
z=x+y
# Converting it to a callable object
# so that it takes matrix as parameters
f = function([x, y], z)
f(5, 7)
```

4. Test program for PyTorch## The usual imports

```
import torch
import torch.nn as nn
 ## print out the pytorch version used
print(torch.__version__)
```

Output of Code:

Note: Run the code and attach your output of the code here.

Conclusion :

Tensorflow , PyTorch,Keras and Theano all these packages are installed and ready for Deep learning applications . As per application domain and dataset we can choose the appropriate package and build required type of Neural Network.

ASSIGNMENT NO.2

Title: Implementing Feedforward neural networks with Keras and TensorFlow.

a. Import the necessary packages

b. Load the training and testing data (MNIST/CIFAR10)

c. Define the network architecture using Keras

d. Train the model using SGD

e. Evaluate the network

f. Plot the training loss and accuracy

Theory :

Feed Forward Neural Network's place within the universe of Machine Learning

A visual explanation of how Feed Forward NNs work

- Network structure and terminology
- Parameters and activation functions
- Loss functions, optimizers, and training

Python examples of how to build and train your own Feed Forward Neural Networks TensorFlow is an open-source platform for machine learning. Keras is the high-level application programming interface (API) of TensorFlow. Using Keras, you can rapidly develop a prototype system and test it out. This is the first in a three-part series on using TensorFlow for supervised classification tasks. Note: You can run the code shown in this tutorial in Google Colab or download the Python notebook here to run it locally.

STEPS

1. System requirements

Ubuntu 16.04 or higher (64-bit)

2. Install Miniconda

curl https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -o Miniconda3-latest-Linux-x86_64.sh bash Miniconda3-latest-Linux-x86_64.sh

3. Create a conda environment

1. conda create --name tf python=3.9

2. To Activate: conda deactivate

3. To Deactivate: conda activate tf

4. GPU setup

1. nvidia-smi

2. conda install -c conda-forge cudatoolkit=11.2 cudnn=8.1.0

3. export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$CONDA_PREFIX/lib/

4. mkdir -p $CONDA_PREFIX/etc/conda/activate.d

5. echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$CONDA_PREFIX/lib/' > $CONDA_PREFIX/etc/conda/activate.d/env_vars.sh

5. Install TensorFlow

1. pip install --upgrade pip

2. pip install tensorflow6. Verify install

1. python3 -c "import tensorflow as tf; print(tf.reduce_sum(tf.random.normal([1000, 1000])))" Verify the GPU setup:

2. python3 -c "import tensorflow as tf; print(tf.config.list_physical_devices('GPU'))" How to install jupyter notebook

3. Install the classic Jupyter Notebook with:

pip install notebook

To run the notebook:

jupyter notebook

How to install packages:

Commands:

1. pip install numpy

2. pip install sklearn

3. pip install matplotlib

4. pip install pandas

5. pip install keras

Conclusion :

By, Implementing Feedforward neural networks with Keras and TensorFlow. As per application

domain and dataset we can choose the appropriate package and build required type of Neural Network.

ASSIGNMENT NO.3

Title: Build the Image classification model by dividing the model into following 4 stages: a. Loading and preprocessing the image data

b. Defining the model's architecture

c. Training the model

d. Estimating the model's performance

Theory :

Setting up the Structure of our Image Data:

Our data needs to be in a particular format in order to solve an image classification problem. We will see this in action in a couple of sections but just keep these pointers in mind till we get there.

You should have 2 folders, one for the train set and the other for the test set. In the training set, you will have a .csv file and an image folder:

The .csv file contains the names of all the training images and their corresponding true labels The image folder has all the training images.

The .csv file in our test set is different from the one present in the training set. This test set .csv file contains the names of all the test images, but they do not have any corresponding labels. Can you guess why? Ourmodel will be trained on the images present in the training set and the label predictions will happen on the testing set images

If your data is not in the format described above, you will need to convert it accordingly

Before we deep dive into the Python code, let's take a moment to understand how an image classification model is typically designed. We can divide this process broadly into 4 stages. Each stage requires a certain amount of time to execute:

1. Loading and pre-processing Data – 30% time
2. Defining Model architecture – 10% time
3. Training the model – 50% time
4. Estimation of performance – 10% time

Let me explain each of the above steps in a bit more detail. This section is crucial because not every model is built in the first go. You will need to go back after each iteration, fine-tune your steps, and run it again. Having a solid understanding of the underlying concepts will go a long way in accelerating the entire process.

Stage 1: Loading and pre-processing the data

Data is gold as far as deep learning models are concerned. Your image classification model has a far better chance of performing well if you have a good amount of images in the training set. Also, the shape of the data varies according to the architecture/framework that we use.

Hence, the critical data pre-processing step (the eternally important step in any project). I highly recommend going through the 'Basics of Image Processing in Python' to understand more about how pre-processing works with image data.

But we are not quite there yet. In order to see how our model performs on unseen data (and before exposing it to the test set), we need to create a validation set. This is done by partitioning the training set data.

In short, we train the model on the training data and validate it on the validation data. Once we are satisfied with the model's performance on the validation set, we can use it for making predictions on the test data.

Time required for this step: We require around **2-3 minutes** for this task.

Stage 2: Defining the model's architecture

This is another crucial step in our deep learning model building process. We have to define how our model will look and that requires answering questions like:

- How many convolutional layers do we want?
- What should be the activation function for each layer?
- How many hidden units should each layer have?

And many more. These are essentially the hyperparameters of the model which play a MASSIVE part in deciding how good the predictions will be.

How do we decide these values? Excellent question! A good idea is to pick these values based on existing research/studies. Another idea is to keep experimenting with the values until you find the best match but this can be quite a time consuming process.

Time required for this step: It should take around **1 minute** to define the architecture of the model.

Stage 3: Training the model

For training the model, we require:

- Training images and their corresponding true labels
- Validation images and their corresponding true labels (we use these labels only to validate the model and not during the training phase)

We also define the number of epochs in this step. For starters, we will run the model for 10 epochs (you can change the number of epochs later).

Time required for this step: Since training requires the model to learn structures, we need around **5 minutes** to go through this step.

And now time to make predictions!

Stage 4: Estimating the model's performance

Finally, we load the test data (images) and go through the pre-processing step here as well. We then predict the classes for these images using the trained model.

Time required for this step: ~ 1 minute.

Conclusion :

referel link https://www.analyticsvidhya.com/blog/2019/01/build-image-classification-model10-minutes/ to Build the Image classification model.

ASSIGNMENT NO.4

Title: Use Autoencoder to implement anomaly detection. Build the model by using: a. Import required libraries

b. Upload / access the dataset

c. Encoder converts it into latent representation

d. Decoder networks convert it back to the original input

e. Compile the models with Optimizer, Loss, and Evaluation Metrics

Theory :

- What is Autoencoder?
- Applications of Autoencoders
- Architecture of Autoencoder.
- Different types of autoencoders
- Simple autoencoders
- Sparse autoencoders
- Deep autoencoders
- Convolutional autoencoders
- Denoising autoencoders
- Variational autoencoders
- Advantages of Autoencoders
- How autoencoders can be used for Anomaly Detection?
- Anomaly Detection in Cardio dataset using tensorflow.

APPLICATION OF AUTOENCODERS:

Anomaly Detection: Autoencoders use the property of a neural network in a special way to accomplish some efficient methods of training networks to learn normal behavior. When an outlier data point arrives, the auto-encoder cannot codify it well. It learned to represent patterns not existing in this data. When trying to reconstruct the original data from its compact representation, the reconstruction will not resemble the original data. Thus helping to detect anomalies when they occur. The goal of such a process is to try to reconstruct the original input from the encoded data, which is critical in building an anomaly detection module.

Dimensionality Reduction: It can learn non-linear transformations, with a non-linear activation function and multiple layers. The objective of an autoencoder is to learn a compressed, distributed representation for the given data, which we can use for the purpose of dimensionality reduction.

Feature Extraction: Here, while in the process of reducing construction error, Encoding part of Autoencoders helps to learn important hidden features present in the input data. By this, model generates, a new set of combination of original features.

Sequence to sequence prediction: LSTMs-based autoencoders uses the Encoder-Decoder Model that can capture temporal structure of a sentence in Machine Translation process.

Recommendation system: We can use Deep Autoencoders to understand user preferences to recommend movies, books or other items. Following are the steps to build recommender system using autoencoder:

The input data is the clustering of similar users based on interests.Interests of users are categorized by Videos Watched, Watched Time for each video etc,,,

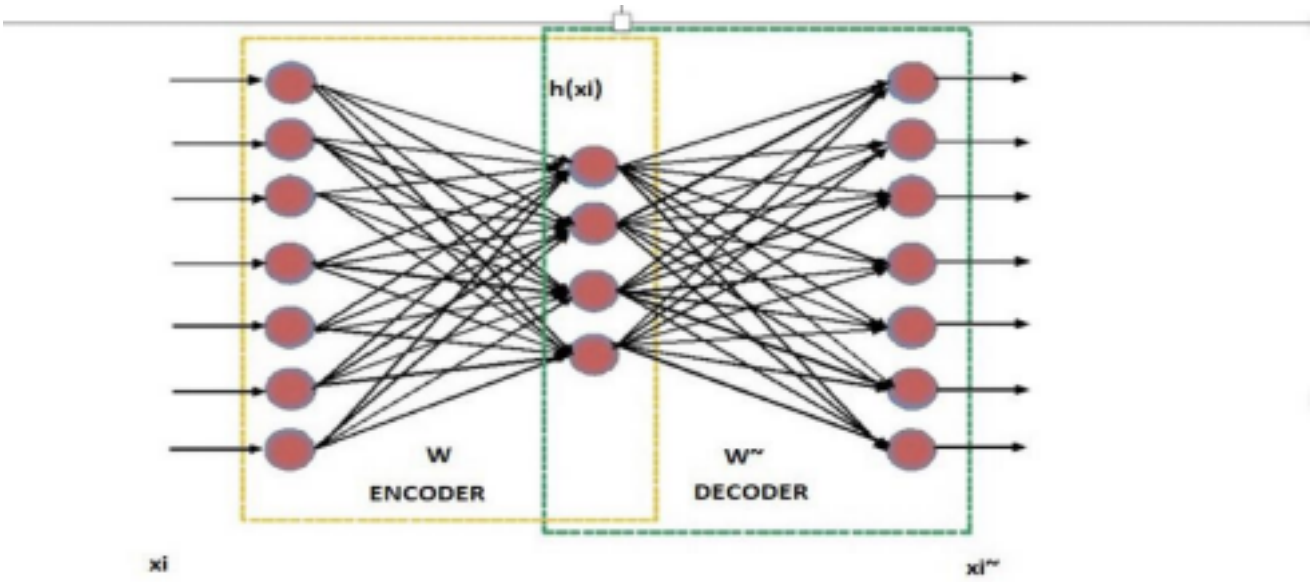
A cluster can be created with similar kind of data from the above filter.Encoder part will capture the interests of the user Decoder part will try to project the interests on two parts:Existing unseen content New content from content creators

WHAT IS AUTOENCODER?

- Autoencoders are neural networks that have the ability to discover low-dimensional representations of high-dimensional data. From this, it should be able to reconstruct the input from the output. There are 3 major parts in an Autoencoder Architecture, as below:
- An Encoder, which reduces the dimensionality of a high dimensional dataset to a low dimensional one.
- Code, which contains the reduced representation of the input that is fed into the decoder. ● A Decoder which expands the low-dimensional data to high-dimensional data. ● Here, compression and decompression functions are
    - Data-Specific, which means the model will be able to compress those data on which it was

trained. This feature of the model helps to compress the majority class of an imbalanced dataset.

- Lossy, which means which means that the decompressed outputs gets degraded compared to the original inputs.
- Learned Automatically from examples, which means it works better with a specific type of input.



Conclusion :

referel link https://towardsdatascience.com/anomaly-detection-using-autoencoders-5b032178a1ea/ to Use Autoencoder to implement anomaly detection.

ASSIGNMENT NO.5

Title: Implement the Continuous Bag of Words (CBOW) Model. Stages can be: a. Data preparation

b. Generate training data

c. Train model

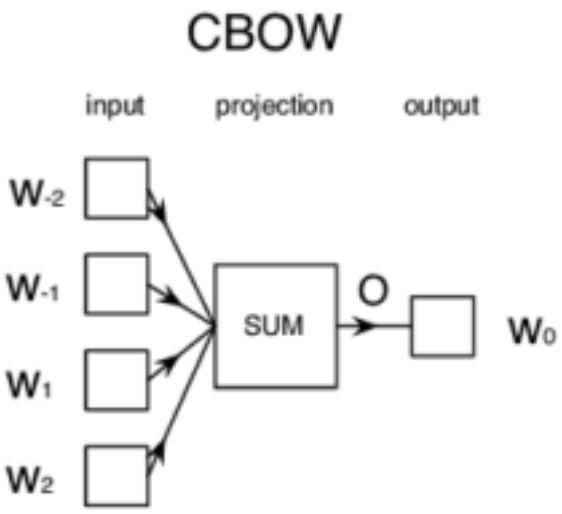
d. Output

Theory:

What is the CBOW Model?

The CBOW model tries to understand the context of the words and takes this as input. It then tries to predict words that are contextually accurate. Let us consider an example for understanding this. Consider the sentence: 'It is a pleasant day' and the word 'pleasant' goes as input to the neural network. We are trying to predict the word 'day' here. We will use the one-hot encoding for the input words and measure the error rates with the one-hot encoded target word. Doing this will help us predict the output based on the word with least error.

Model Architecture



The CBOW model architecture is as shown above. The model tries to predict the target word by trying to understand the context of the surrounding words. Consider the same sentence as above, 'It

is a pleasant day'.The model converts this sentence into word pairs in the form (contextword, targetword). The user will have to set the window size. If the window for the context word is 2 then the word pairs would look like this: ([it, a], is), ([is, pleasant], a),([a, day], pleasant). With these word pairs, the model tries to predict the target word considered the context words.

If we have 4 context words used for predicting one target word the input layer will be in the form of four 1XW input vectors. These input vectors will be passed to the hidden layer where it is multiplied by a WXN matrix. Finally, the 1XN output from the hidden layer enters the sum layer where an element-wise summation is performed on the vectors before a final activation is performed and the output is obtained.

Implementation of the CBOW Model

For the implementation of this model, we will use a sample text data about coronavirus. You can use any text data of your choice. But to use the data sample I have used click here to download the data.

Now that you have the data ready, let us import the libraries and read our dataset.

```
import numpy as np
import keras.backend as K
from keras.models import Sequential
from keras.layers import Dense, Embedding, Lambda
from keras.utils import np_utils
from keras.preprocessing import sequence
from keras.preprocessing.text import Tokenizer
import gensim
data=open('/content/gdrive/My Drive/covid.txt','r')
corona_data = [text for text in data if text.count(' ') >= 2]
vectorize = Tokenizer()
vectorize.fit_on_texts(corona_data)
corona_data = vectorize.texts_to_sequences(corona_data)
total_vocab = sum(len(s) for s in corona_data)
word_count = len(vectorize.word_index) + 1

window_size = 2
```

In the above code, I have also used the built-in method to tokenize every word in the dataset and fit our data to the tokenizer. Once that is done, we need to calculate the total number of words and the total number of sentences as well for further use. As mentioned in the model architecture, we need to assign the window size and I have assigned it to 2.

The next step is to write a function that generates pairs of the context words and the target words. The function below does exactly that. Here we have generated a function that takes in window sizes separately for target and the context and creates the pairs of contextual words and target words.

```
def cbow_model(data, window_size, total_vocab):
    total_length = window_size*2
```

```
    for text in data:
        text_len = len(text)
        for idx, word in enumerate(text):
            context_word = []
            target = []
            begin = idx - window_size
            end = idx + window_size + 1
            context_word.append([text[i] for i in range(begin, end) if 0 <= i < text_len and i != idx])
            target.append(word)
            contextual = sequence.pad_sequences(context_word, total_length=total_length)
            final_target = np_utils.to_categorical(target, total_vocab)
            yield(contextual, final_target)
```

Finally, it is time to build the neural network model that will train the CBOW on our sample data.

```
model = Sequential()
model.add(Embedding(input_dim=total_vocab, output_dim=100,
input_length=window_size*2)) model.add(Lambda(lambda x: K.mean(x, axis=1),
output_shape=(100,)))
model.add(Dense(total_vocab, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')

for i in range(10):
    cost = 0
    for x, y in cbow_model(data, window_size, total_vocab):
        cost += model.train_on_batch(contextual, final_target)
    print(i, cost)
```

Once we have completed the training its time to see how the model has performed and test it on some words. But to do this, we need to create a file that contains all the vectors. Later we can access these vectors using the gensim library.

```
dimensions=100
vect_file = open('/content/gdrive/My Drive/vectors.txt' ,'w')
```

```
vect_file.write('{} {}\n'.format(total_vocab,dimensions))
```

Next, we will access the weights of the trained model and write it to the above created file.

```
weights = model.get_weights()[0]
for text, i in vectorize.word_index.items():
    final_vec = ' '.join(map(str, list(weights[i, :])))
    vect_file.write('{} {}\n'.format(text, final_vec))
vect_file.close()
```

Now we will use the vectors that were created and use them in the gensim model. The word I have chosen in 'virus'.

```
cbow_output = gensim.models.KeyedVectors.load_word2vec_format('/content/gdrive/My Drive/vectors.txt', binary=False)
cbow_output.most_similar(positive=['virus'])
```

The output shows the words that are most similar to the word 'virus' along with the sequence or degree of similarity. The words like symptoms and incubation are contextually very accurate with the word virus which proves that CBOW model successfully understands the context of the data.

Conclusion

we saw what a CBOW model is and how it works. We also implemented the model on a custom dataset and got good output. The purpose here was to give you a high-level idea of what word embeddings are and how CBOW is useful. These can be used for text recognition, speech to text conversion etc.

ASSIGNMENT NO.6

Title: Object detection using Transfer Learning of CNN architectures. a. Load in a pre-trained CNN model trained on a large dataset

b. Freeze parameters (weights) in model's lower convolutional layers

c. Add custom classifier with several layers of trainable parameters to model

d. Train classifier layers on training data available for task

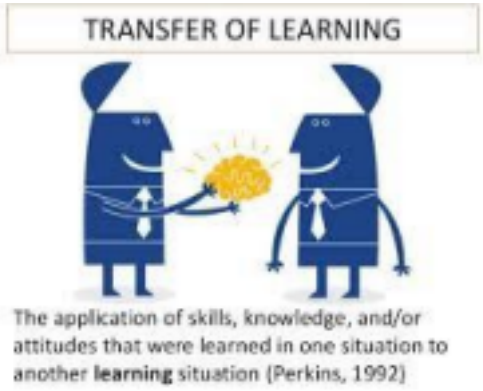
e. Fine-tune hyper parameters and unfreeze more layers as needed.

Theory :

What is transfer learning?

Let us start with developing an intuition for transfer learning. Let us understand from a simple teacher – student analogy.

A teacher has years of experience in the particular topic he/she teaches. With all this accumulated information, the lectures that students get is a concise and brief overview of the topic. So it can be seen as a "transfer" of information from the learned to a novice.



What is a Pre-trained Model?

Simply put, a pre-trained model is a model created by some one else to solve a similar problem. Instead of building a model from scratch to solve a similar problem, you use the model trained on other problem as a starting point.

For example, if you want to build a self learning car. You can spend years to build a decent image recognition algorithm from scratch or you can take inception model (a pre-trained model) from Google which was built on ImageNet data to identify images in those pictures.

A pre-trained model may not be 100% accurate in your application, but it saves huge efforts required to re-invent the wheel. Let me show this to you with a recent example.

Why would we use Pre-trained Models?

I spent my last week working on a problem at CrowdAnalytix platform – Identifying themes from mobile case images. This was an image classification problem where we were given 4591 images in

the training dataset and 1200 images in the test dataset. The objective was to classify the images into one of the 16 categories. After the basic pre-processing steps, I started off with a simple MLP model with the following architecture

Layer (type)	Output Shape	Param #
flatten_4 (Flatten)	(None, 150528)	0
dense_9 (Dense)	(None, 500)	75264500
activation_9 (Activation)	(None, 500)	0
dropout_7 (Dropout)	(None, 500)	0
dense_10 (Dense)	(None, 500)	250500
activation_10 (Activation)	(None, 500)	0
dropout_8 (Dropout)	(None, 500)	0
dense_11 (Dense)	(None, 500)	250500
activation_11 (Activation)	(None, 500)	0
dropout_9 (Dropout)	(None, 500)	0
dense_12 (Dense)	(None, 16)	8016
activation_12 (Activation)	(None, 16)	0

Total params: 75,773,516
Trainable params: 75,773,516
Non-trainable params: 0

To simplify the above architecture after flattening the input image [224 X 224 X 3] into [150528], I used three hidden layers with 500, 500 and 500 neurons respectively. The output layer had 16 neurons which correspond to the number of categories in which we need to classify the input image.

I barely managed a training accuracy of 6.8 % which turned out to be very bad. Even experimenting with hidden layers, number of neurons in hidden layers and drop out rates. I could not manage to substantially increase my training accuracy. Increasing the hidden layers and the number of neurons, caused 20 seconds to run a single epoch on my Titan X GPU with 12 GB VRAM.

Below is an output of the training using the MLP model with the above architecture.

Epoch 10/10

50/50 [==============================] – 21s – loss: 15.0100 – acc: 0.0688

As, you can see MLP was not going to give me any better results without exponentially increasing my training time. So I switched to Convolutional Neural Network to see how they perform on this dataset and whether I would be able to increase my training accuracy.

The CNN had the below architecture –

Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 220, 220, 32)	2432
activation_27 (Activation)	(None, 220, 220, 32)	0
max_pooling2d_7 (MaxPooling2	(None, 55, 55, 32)	0
conv2d_8 (Conv2D)	(None, 51, 51, 32)	25632
activation_28 (Activation)	(None, 51, 51, 32)	0
max_pooling2d_8 (MaxPooling2	(None, 12, 12, 32)	0
conv2d_9 (Conv2D)	(None, 8, 8, 64)	51264
activation_29 (Activation)	(None, 8, 8, 64)	0
max_pooling2d_9 (MaxPooling2	(None, 2, 2, 64)	0
flatten_8 (Flatten)	(None, 256)	0
dense_21 (Dense)	(None, 64)	16448
activation_30 (Activation)	(None, 64)	0
dropout_12 (Dropout)	(None, 64)	0
dense_22 (Dense)	(None, 16)	1040
activation_31 (Activation)	(None, 16)	0

Total params: 96,816
Trainable params: 96,816
Non-trainable params: 0

I used 3 convolutional blocks with each block following the below architecture 32 filters of size 5 X 5
Activation function – relu
Max pooling layer of size 4 X 4

The result obtained after the final convolutional block was flattened into a size [256] and passed into a single hidden layer of with 64 neurons. The output of the hidden layer was passed onto the output layer after a drop out rate of 0.5.

The result obtained with the above architecture is summarized below

Epoch 10/10

50/50 [==============================] – 21s – loss: 13.5733 – acc: 0.1575

Though my accuracy increased in comparison to the MLP output, it also increased the time taken to run a single epoch – 21 seconds.

But the major point to note was that the majority class in the dataset was around 17.6%. So, even if we had predicted the class of every image in the train dataset to be the majority class, we would have performed better than MLP and CNN respectively. Addition of more convolutional blocks substantially increased my training time. This led me to switch onto using pre-trained models where I would not have to train my entire architecture but only a few layers.

So, I used VGG16 model which is pre-trained on the ImageNet dataset and provided in the keras library for use. Below is the architecture of the VGG16 model which I used.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000
dense_1 (Dense)	(None, 16)	16016

Total params: 138,373,560
Trainable params: 138,373,560
Non-trainable params: 0

The only change that I made to the VGG16 existing architecture is changing the softmax layer with 1000 outputs to 16 categories suitable for our problem and re-training the dense layer.

This architecture gave me an accuracy of 70% much better than MLP and CNN. Also, the biggest benefit of using the VGG16 pre-trained model was almost negligible time to train the dense layer

with greater accuracy.

So, I moved forward with this approach of using a pre-trained model and the next step was to fine tune my VGG16 model to suit this problem.

How can I use Pre-trained Models?

What is our objective when we train a neural network? We wish to identify the correct weights for the network by multiple forward and backward iterations. By using pre-trained models which have been previously trained on large datasets, we can directly use the weights and architecture obtained and apply the learning on our problem statement. This is known as transfer learning. We "transfer the learning" of the pre-trained model to our specific problem statement.

You should be very careful while choosing what pre-trained model you should use in your case. If the problem statement we have at hand is very different from the one on which the pre-trained model was trained – the prediction we would get would be very inaccurate. For example, a model previously trained for speech recognition would work horribly if we try to use it to identify objects using it.

We are lucky that many pre-trained architectures are directly available for us in the Keras library. Imagenet data set has been widely used to build various architectures since it is large enough (1.2M images) to create a generalized model. The problem statement is to train a model that can correctly classify the images into 1,000 separate object categories. These 1,000 image categories represent object classes that we come across in our day-to-day lives, such as species of dogs, cats, various household objects, vehicle types etc.

These pre-trained networks demonstrate a strong ability to generalize to images outside the ImageNet dataset via transfer learning. We make modifications in the pre-existing model by fine tuning the model. Since we assume that the pre-trained network has been trained quite well, we would not want to modify the weights too soon and too much. While modifying we generally use a learning rate smaller than the one used for initially training the model.

Ways to Fine tune the model

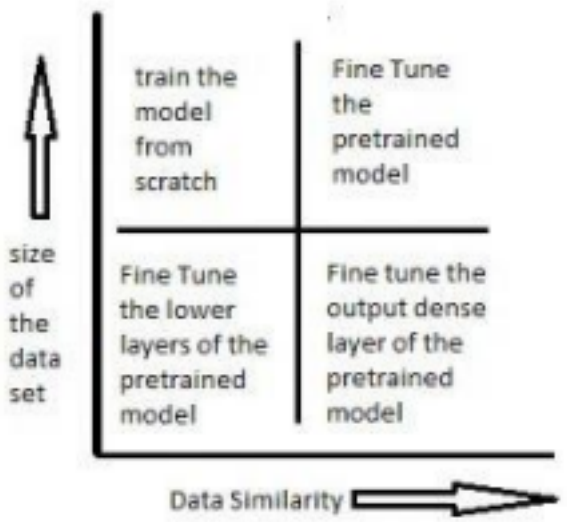
Feature extraction – We can use a pre-trained model as a feature extraction mechanism. What we

can do is that we can remove the output layer(the one which gives the probabilities for being in each of the 1000 classes) and then use the entire network as a fixed feature extractor for the new data set.

Use the Architecture of the pre-trained model – What we can do is that we use architecture of the model while we initialize all the weights randomly and train the model according to our dataset again.

Train some layers while freeze others – Another way to use a pre-trained model is to train is partially. What we can do is we keep the weights of initial layers of the model frozen while we retrain only the higher layers. We can try and test as to how many layers to be frozen and how many to be trained.

The below diagram should help you decide on how to proceed on using the pre trained model in your case –



Use the pre-trained models to identify handwritten digits

Let's now try to use a pretrained model for a simple problem. There are various architectures that have been trained on the imageNet data set. You can go through various architectures here. I have used vgg16 as pretrained model architecture and have tried to identify handwritten digits using it. Let's see in which of the above scenarios would this problem fall into. We have around 60,000 training images of handwritten digits. This data set is definitely small. So the situation would either fall into scenario 1 or scenario 2. We shall try to solve the problem using both these scenarios. The

data set can be downloaded from here.

Retrain the output dense layers only – Here we use vgg16 as a feature extractor. We then use these features and send them to dense layers which are trained according to our data set. The output layer is also replaced with our new softmax layer relevant to our problem. The output layer in a vgg16 is a softmax activation with 1000 categories. We remove this layer and replace it with a softmax layer of 10 categories. We just train the weights of these layers and try to identify the digits.

 # importing required libraries

```
from keras.models import Sequential
from scipy.misc import imread
get_ipython().magic('matplotlib inline')
import matplotlib.pyplot as plt
import numpy as np
import keras
from keras.layers import Dense
import pandas as pd

from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
import numpy as np
from keras.applications.vgg16 import decode_predictions
train=pd.read_csv("R/Data/Train/train.csv")
test=pd.read_csv("R/Data/test.csv")

train_path="R/Data/Train/Images/train/"
test_path="R/Data/Train/Images/test/"

from scipy.misc import imresize
```

 # preparing the train dataset

```
train_img=[]
for i in range(len(train)):
```

```
    temp_img=image.load_img(train_path+train['filename'][i],target_size=(224,224))

    temp_img=image.img_to_array(temp_img)

    train_img.append(temp_img)

#converting train images to array and applying mean subtraction processing

train_img=np.array(train_img)
train_img=preprocess_input(train_img)
# applying the same procedure with the test dataset

test_img=[]
for i in range(len(test)):

    temp_img=image.load_img(test_path+test['filename'][i],target_size=(224,224))

    temp_img=image.img_to_array(temp_img)

    test_img.append(temp_img)

test_img=np.array(test_img)
test_img=preprocess_input(test_img)

# loading VGG16 model weights
model = VGG16(weights='imagenet', include_top=False)
# Extracting features from the train dataset using the VGG16 pre-trained model

features_train=model.predict(train_img)

# Extracting features from the train dataset using the VGG16 pre-trained model

features_test=model.predict(test_img)

# flattening the layers to conform to MLP input

train_x=features_train.reshape(49000,25088)
```

```
# converting target variable to array

train_y=np.asarray(train['label'])
# performing one-hot encoding for the target variable

train_y=pd.get_dummies(train_y)
train_y=np.array(train_y)
# creating training and validation set

from sklearn.model_selection import train_test_split
X_train, X_valid, Y_train, Y_valid=train_test_split(train_x,train_y,test_size=0.3, random_state=42)

# creating a mlp model
from keras.layers import Dense, Activation
model=Sequential()

model.add(Dense(1000, input_dim=25088, activation='relu',kernel_initializer='uniform'))
keras.layers.core.Dropout(0.3, noise_shape=None, seed=None)

model.add(Dense(500,input_dim=1000,activation='sigmoid'))
keras.layers.core.Dropout(0.4, noise_shape=None, seed=None)

model.add(Dense(150,input_dim=500,activation='sigmoid'))
keras.layers.core.Dropout(0.2, noise_shape=None, seed=None)

model.add(Dense(units=10))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy', optimizer="adam", metrics=['accuracy'])

# fitting the model

model.fit(X_train, Y_train, epochs=20, batch_size=128,validation_data=(X_valid,Y_valid))
```

2. Freeze the weights of first few layers – Here what we do is we freeze the weights of the first 8 layers of the vgg16 network, while we retrain the subsequent layers. This is because the first few

layers capture universal features like curves and edges that are also relevant to our new problem. We want to keep those weights intact and we will get the network to focus on learning dataset specific features in the subsequent layers.

Code for freezing the weights of first few layers.

```
from keras.models import Sequential
from scipy.misc import imread
get_ipython().magic('matplotlib inline')
import matplotlib.pyplot as plt
import numpy as np
import keras
from keras.layers import Dense
import pandas as pd

from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
import numpy as np

 from keras.applications.vgg16 import decode_predictions
from keras.utils.np_utils import to_categorical

from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.optimizers import SGD
from keras.layers import Input, Dense, Convolution2D, MaxPooling2D, AveragePooling2D,
ZeroPadding2D, Dropout, Flatten, merge, Reshape, Activation

from sklearn.metrics import log_loss

train=pd.read_csv("R/Data/Train/train.csv")
test=pd.read_csv("R/Data/test.csv")
train_path="R/Data/Train/Images/train/"
test_path="R/Data/Train/Images/test/"
```

```
from scipy.misc import imresize

train_img=[]
for i in range(len(train)):

    temp_img=image.load_img(train_path+train['filename'][i],target_size=(224,224))

    temp_img=image.img_to_array(temp_img)

    train_img.append(temp_img)

train_img=np.array(train_img)
train_img=preprocess_input(train_img)

test_img=[]
for i in range(len(test)):

temp_img=image.load_img(test_path+test['filename'][i],target_size=(224,224))

    temp_img=image.img_to_array(temp_img)

    test_img.append(temp_img)

test_img=np.array(test_img)
test_img=preprocess_input(test_img)

from keras.models import Model

def vgg16_model(img_rows, img_cols, channel=1,

    num_classes=None): model = VGG16(weights='imagenet',

    include_top=True)

    model.layers.pop()

    model.outputs = [model.layers[-1].output]
```

```
    model.layers[-1].outbound_nodes = []

        x=Dense(num_classes, activation='softmax')(model.output)

    model=Model(model.input,x)

#To set the first 8 layers to non-trainable (weights will not be updated)

        for layer in model.layers[:8]:

      layer.trainable = False

# Learning rate is changed to 0.001
    sgd = SGD(lr=1e-3, decay=1e-6, momentum=0.9, nesterov=True)
    model.compile(optimizer=sgd, loss='categorical_crossentropy',

    metrics=['accuracy']) return model

train_y=np.asarray(train['label'])

le = LabelEncoder()

train_y = le.fit_transform(train_y)

train_y=to_categorical(train_y)

train_y=np.array(train_y)

from sklearn.model_selection import train_test_split
X_train, X_valid, Y_train, Y_valid=train_test_split(train_img,train_y,test_size=0.2, random_state=42)

# Example to fine-tune on 3000 samples from Cifar10

img_rows, img_cols = 224, 224 # Resolution of inputs
channel = 3
num_classes = 10
batch_size = 16
```

nb_epoch = 10

 # Load our model

model = vgg16_model(img_rows, img_cols, channel, num_classes)

model.summary()

Start Fine-tuning

model.fit(X_train,

Y_train,batch_size=batch_size,epochs=nb_epoch,shuffle=True,verbose=1,validation_data=(X_valid, Y_valid))

Make predictions

predictions_valid = model.predict(X_valid, batch_size=batch_size, verbose=1)

Cross-entropy loss score

score = log_loss(Y_valid, predictions_valid)

Conclusion :

referel link https://www.analyticsvidhya.com/blog/2017/06/transfer-learning-the-art-of-fine tuning-a-pre-trained-model/ to Use Autoencoder to implement anomaly detection.

1. Update Operation

```
> db.Employee.update({_id:66},{$set:{Name:"Ramanand"}},{multi:true})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

This command is used to update the document values. (Primary key is specified hence it will update single document)

```
> db.Employee.update({Name:"XYZ"},{$set:{Name:"Niya"}},{multi:true})
WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 })
```

Above command has updated two documents.

Conclusion:

In this assignment, we learned about difference between relational and NoSQL databases, installation of MongoDB, creating and accessing database in MongoD